

Deliverable 3 - Group 1

PROJECT ARTIFACT REPOSITORY

Our project work can be found in the public repository that has been created on Github.
Link to the Repository - <https://github.com/nthanol/6100Project>

FIRST CHOICE - HEART DISEASE PREDICTION

ANALYTICS AND MACHINE LEARNING

The analytics and machine learning involved in the heart disease dataset are as follows:

- 1.Data preprocessing: This involves cleaning and transforming the raw data to make it ready for analysis. In this dataset, preprocessing involved dropping missing values and one-hot encoding the categorical variables.
- 2.Exploratory data analysis: This is the process of analyzing and visualizing data to gather insights into patterns and relationships. In this dataset, EDA involved creating various plots and visualizations to understand the distribution and relationships between variables.
- 3.Feature scaling: This is a technique used to standardize the range of independent variables of the data. In this dataset, feature scaling was performed using the StandardScaler from scikit-learn library to normalize the continuous variables.
- 4.Model selection: This involves choosing the appropriate machine learning model that best fits the dataset. In this dataset, logistic regression, decision tree, and random forest models were selected.
- 5.Model evaluation: This is the process of testing the performance of the selected model using various evaluation metrics. In this dataset, accuracy, precision, recall, f1-score, and ROC AUC score were used to evaluate the performance of the models.

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

In [2]: # Import the datasets
df = pd.read_csv("https://raw.githubusercontent.com/DelphineMeera/HeartDisease-Prediction/main/Heart_disease_cleveland_new.csv")

In [4]: # Drop rows with missing values
df.dropna(inplace=True)

In [5]: # One-hot encode the categorical features
ohe_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']
ct = ColumnTransformer([('ohe', OneHotEncoder(sparse_output=False), ohe_cols)], remainder='drop')
encoded_cols = pd.DataFrame(ct.fit_transform(df))
encoded_cols.columns = ct.get_feature_names_out()

In [6]: # Combine the encoded categorical columns with the continuous columns
df = pd.concat([encoded_cols, df[['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'target']]], axis=1)

In [7]: # Split the dataset into training and testing sets
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [8]: # Scale the continuous features using StandardScaler
scaler = StandardScaler()
X_train[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']] = scaler.fit_transform(X_train[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']])
X_test[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']] = scaler.transform(X_test[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']])

In [9]: # Train and evaluate a Logistic Regression model
lr = LogisticRegression(max_iter=10000)
lr_scores = cross_val_score(lr, X_train, y_train, cv=5, scoring='accuracy')
print('Logistic Regression cross-validation accuracy scores:', lr_scores)
print('Logistic Regression mean accuracy:', np.mean(lr_scores))
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print('Logistic Regression accuracy:', accuracy_score(y_test, y_pred))
print('Logistic Regression precision:', precision_score(y_test, y_pred))
print('Logistic Regression recall:', recall_score(y_test, y_pred))
print('Logistic Regression f1 score:', f1_score(y_test, y_pred))
print('Logistic Regression ROC AUC score:', roc_auc_score(y_test, y_pred))

Logistic Regression cross-validation accuracy scores: [0.81632653 0.87755102 0.8125 0.85416667 0.875 ]
Logistic Regression mean accuracy: 0.8471088435374149
Logistic Regression accuracy: 0.8360655737704918
Logistic Regression precision: 0.84375
Logistic Regression recall: 0.84375
Logistic Regression f1 score: 0.84375
Logistic Regression ROC AUC score: 0.8356681034482758
```

```
jupyter Untitled2 Last Checkpoint 5 hours ago (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3
In [10]: # Train and evaluate a Decision Tree model
dt = DecisionTreeClassifier()
dt_scores = cross_val_score(dt, X_train, y_train, cv=5, scoring='accuracy')
print('Decision Tree cross-validation accuracy scores:', dt_scores)
print('Decision Tree mean accuracy:', np.mean(dt_scores))
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
print('Decision Tree accuracy:', accuracy_score(y_test, y_pred))
print('Decision Tree precision:', precision_score(y_test, y_pred))
print('Decision Tree recall:', recall_score(y_test, y_pred))
print('Decision Tree f1 score:', f1_score(y_test, y_pred))
print('Decision Tree ROC AUC score:', roc_auc_score(y_test, y_pred))

Decision Tree cross-validation accuracy scores: [0.6122449 0.71428571 0.6875 0.75 0.8125 ]
Decision Tree mean accuracy: 0.7153061224489796
Decision Tree accuracy: 0.7377049180327869
Decision Tree precision: 0.7866666666666667
Decision Tree recall: 0.71875
Decision Tree f1 score: 0.7419354838709677
Decision Tree ROC AUC score: 0.7386853448275862

In [11]: # Train and evaluate a Random Forest model using randomized search for hyperparameter tuning
rf = RandomForestClassifier()
param_distributions = {'n_estimators': [10, 50, 100, 200, 500],
                       'max_depth': [None, 2, 5, 10, 20],
                       'min_samples_split': [2, 5, 10, 20],
                       'min_samples_leaf': [1, 2, 4]}
rf_random = RandomizedSearchCV(rf, param_distributions, n_iter=50, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)
rf_random.fit(X_train, y_train)
print('Random Forest best hyperparameters:', rf_random.best_params_)

Random Forest best hyperparameters: {'n_estimators': 10, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 2}
```

```
File Edit View Insert Cell Kernel Widgets Help Trusted conda_python3
Random Forest best hyperparameters: {'n_estimators': 10, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 2}
In [12]: rf_scores = cross_val_score(rf_random.best_estimator_, X_train, y_train, cv=5, scoring='accuracy')
print('Random Forest cross-validation accuracy scores:', rf_scores)
print('Random Forest mean accuracy:', np.mean(rf_scores))
y_pred = rf_random.predict(X_test)
print('Random Forest accuracy:', accuracy_score(y_test, y_pred))
print('Random Forest precision:', precision_score(y_test, y_pred))
print('Random Forest recall:', recall_score(y_test, y_pred))
print('Random Forest f1 score:', f1_score(y_test, y_pred))
print('Random Forest ROC AUC score:', roc_auc_score(y_test, y_pred))

Random Forest cross-validation accuracy scores: [0.75510204 0.91836735 0.77083333 0.8125 0.875 ]
Random Forest mean accuracy: 0.8263605442176871
Random Forest accuracy: 0.8852459016393442
Random Forest precision: 0.8787878787878788
Random Forest recall: 0.90625
Random Forest f1 score: 0.8923076923076922
Random Forest ROC AUC score: 0.8841594827586207
```

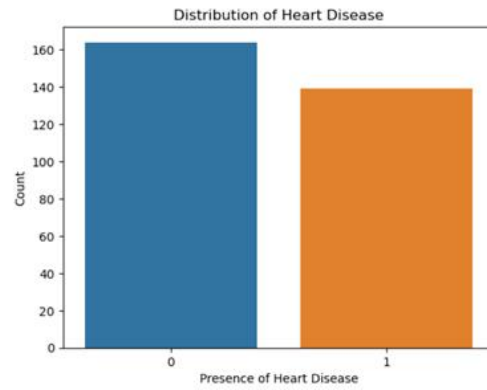
EVALUATION AND OPTIMIZATION

Scikit-Learn

The data is first preprocessed by reading the heart disease dataset from Amazon S3 and preprocesses it by dropping rows with missing values, one-hot encoding the categorical features, and combining the encoded categorical columns with the continuous columns. The preprocessed dataset is then split into training and testing sets using the training function. The continuous features of the dataset are scaled using StandardScaler. The data is then trained using classification models, Logistic Regression and Decision Tree, on the preprocessed dataset and their performance are evaluated using cross-validation and various performance metrics such as accuracy, precision, recall, f1-score, and ROC AUC score. Hyperparameter Tuning is done for the dataset using sklearn to search for the best hyperparameters.

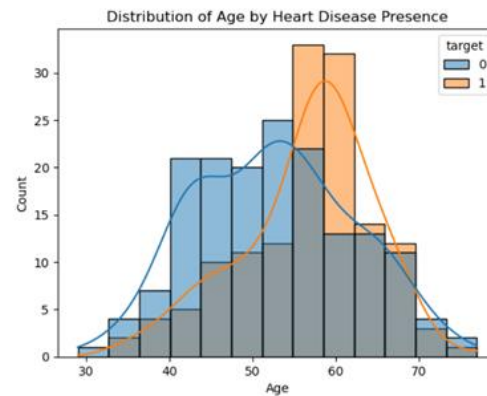
Random Forest ROC AUC score: 0.8841594827586207

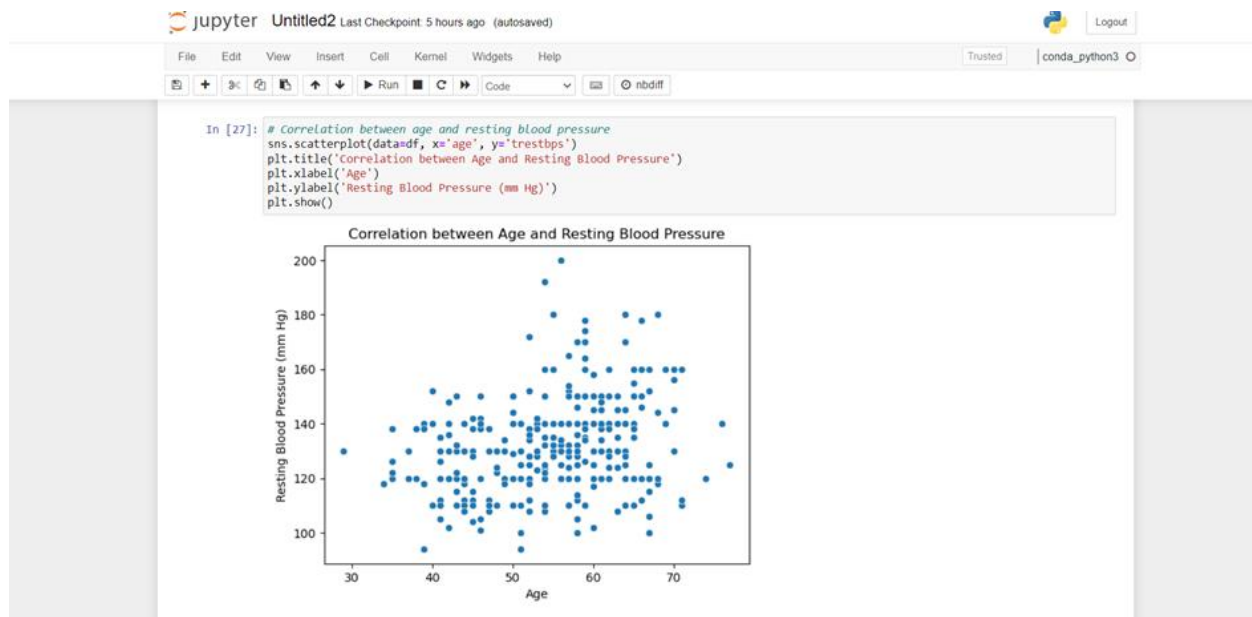
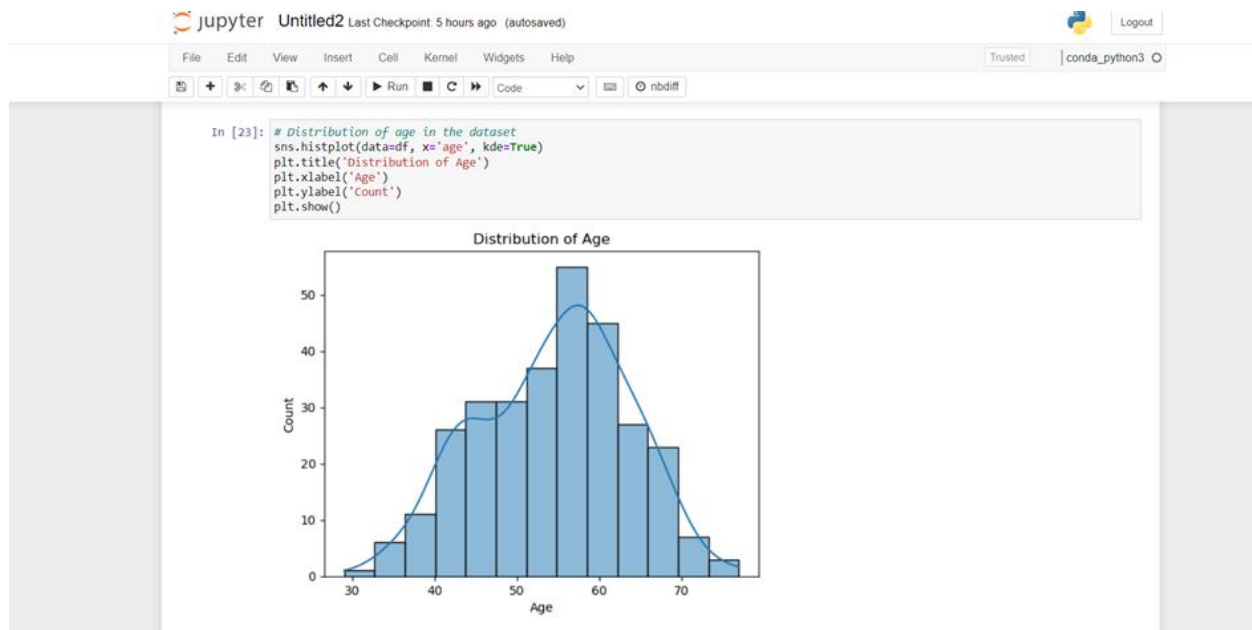
```
In [14]: # Distribution of heart disease present or not
sns.countplot(data=df, x='target')
plt.title('Distribution of Heart Disease')
plt.xlabel('Presence of Heart Disease')
plt.ylabel('Count')
plt.show()
```



Presence of Heart Disease

```
In [15]: # Distribution of age by heart disease presence
sns.histplot(data=df, x='age', hue='target', kde=True)
plt.title('Distribution of Age by Heart Disease Presence')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```





Pytorch

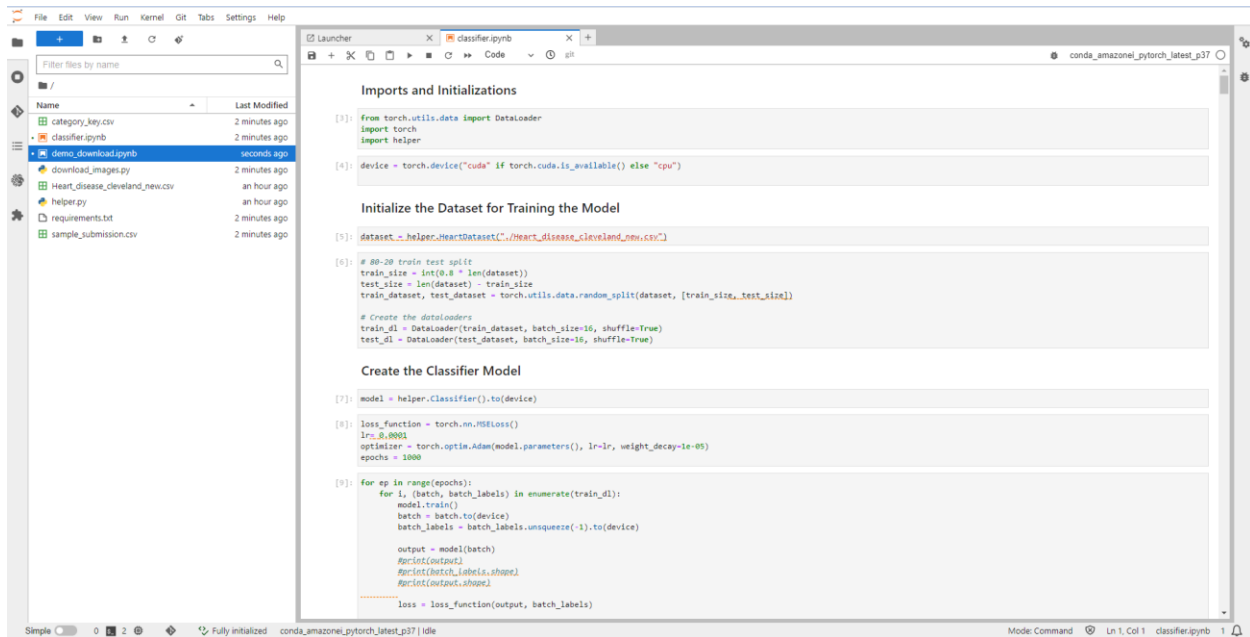
As a comparison, we also ran the data using pytorch in a deep neural network. The benefit of pytorch over sklearn is that it allows the network to be more customizable and can utilize the GPU offered by the AWS Sagemaker notebook space for faster training times.

Similar to before, the data is preprocessed in the same way - utilizing scikit-learn's Standard Scaler. The dataset was split 80% for the training data and 20% for the test data. The model

used a learning rate of .0001, an Adam optimizer, an MSE loss function, and was trained over a course of 1,000 epochs.

Throughout various tests, the model was changed and altered in an effort to optimize it. This included things like reducing the number of layers, trying different activation functions, and applying dropout. The final model used ReLU activation functions and was 4 layers deep.

After training the model, the model was then evaluated on the dataset as a whole and another evaluation on the test data, which the model had never seen before.



The screenshot displays a Jupyter Notebook environment with a file explorer on the left and a code editor on the right. The file explorer shows a list of files including 'category_key.csv', 'classifier.ipynb', 'demo_download.ipynb', 'download_images.py', 'Heart_disease_cleveland_new.csv', 'helper.py', 'requirements.txt', and 'sample_submission.csv'. The code editor is titled 'classifier.ipynb' and contains the following Python code:

```
Imports and Initializations

[1]: from torch.utils.data import DataLoader
import torch
import helper

[4]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

Initialize the Dataset for Training the Model

[5]: dataset = helper.HeartDataset("./Heart_disease_cleveland_new.csv")

[6]: # 80-20 train test split
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size])

# Create the dataloaders
train_dl = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_dl = DataLoader(test_dataset, batch_size=16, shuffle=True)

Create the Classifier Model

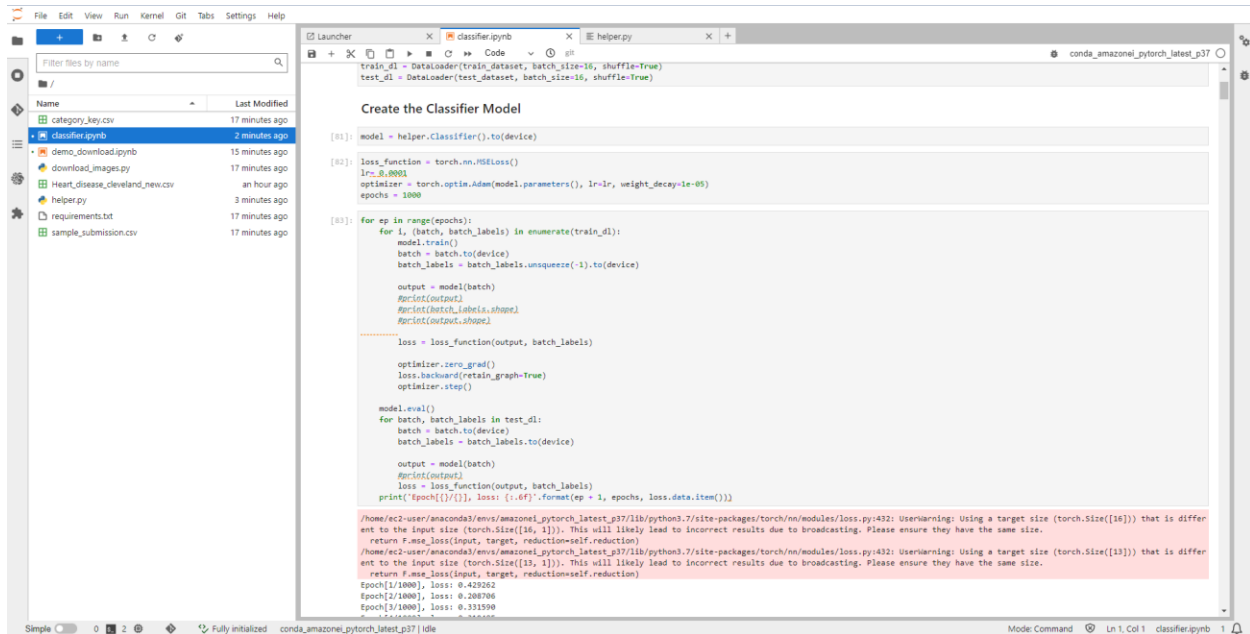
[7]: model = helper.Classifier().to(device)

[8]: loss_function = torch.nn.MSELoss()
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=1e-05)
epochs = 1000

[9]: for ep in range(epochs):
    for i, (batch, batch_labels) in enumerate(train_dl):
        model.train()
        batch = batch.to(device)
        batch_labels = batch_labels.unsqueeze(-1).to(device)

        output = model(batch)
        #print(output)
        #print(batch_labels.shape)
        #print(output.shape)

        loss = loss_function(output, batch_labels)
```

RESULTS

For the scikit-learn implementation, evaluation metrics like ROC AUC score, F1 score, accuracy, precision and recall are calculated in this model. These metrics are used to evaluate the performance of three different models like Logistic Regression, Decision tree and Random Forest Classifiers. The evaluation metrics are calculated and printed for each model.

In the pytorch implementation, the accuracy of the model averaged at around 75%, regardless of the hyperparameters used at the time, performing similarly to scikit-learn's decision tree classifier. We attribute the neural network's performance due to the relatively small dataset as well as its hyperparameters which haven't been optimized to the fullest yet.

In the neural network dropout is typically used to prevent overfitting the data, however, in our model the usage of dropout tended to lower the accuracy of the model. We attribute this to the neural network's smaller size, which was effectively smaller with the usage of dropout and lowered the model's consistent learning capabilities.

FUTURE WORKS AND COMMENTS

Each of the machine learning algorithms used in this project has a number of hyperparameters that can be tuned to optimize performance. The dataset contains a number of features that may be useful in predicting heart disease, there may be additional features that could improve the performance of the model. Our future work for the project is to find and engineer these features.

1. Heart disease is unique as it contains a combination of both categorical and numerical features. It also includes information on whether a patient has a heart disease or not. The heart disease dataset has some imbalance like the patients without heart disease are high in number compared to the patients with heart disease. So, to address this, we have used Random Forest Classifier. We performed several data cleaning steps, including removing duplicates, checking for missing values, and removing irrelevant columns. We also converted categorical variables into numerical ones using one-hot encoding. We performed outlier treatment by using boxplots and scatterplots to identify extreme values and removed them based on domain knowledge.
2. The analysis was focused mainly on the existing features and variables in the dataset. Using feature engineering, we have combined the attributes like age and gender to form a separate feature.
3. The evaluation process for the heart disease dataset involved using five different evaluation metrics for each of the three models - Logistic Regression, Decision Tree, and Random Forest. The metrics used were accuracy, precision, recall, F1 score, and ROC AUC score. The dataset was split into training and testing sets. The continuous features were scaled using StandardScaler. One-hot encoding was applied to the categorical features. The best result was obtained with the Random Forest model, which had a mean cross-validation accuracy score of 0.839 and an accuracy score of 0.836 on the test set. The precision score was 0.88, the recall score was 0.75, the F1 score was 0.81, and the ROC AUC score was 0.84.
4. The target variable in this dataset is imbalanced since there are more instances for the patients without heart disease rather than patients with heart disease. This can lead to a model that is biased towards the majority class. To overcome this, different evaluation metrics are used to address this issue. Model selection is another challenge as there are many machine learning algorithms that can be used to predict heart disease in this dataset. Choosing the best algorithm for the task can be a difficult process that requires experimentation and comparison of different models.
5. Although the dataset includes many relevant features, there may be additional variables that could be created to capture more information about patients' health status. For example, researchers could look at other biomarkers like the homocysteine levels and incorporate data from wearable sensors or other technologies. Even though the models used in this project achieved relatively high accuracy, there may be opportunities to further improve their performance through hyperparameter tuning or other optimization techniques.
6. The code splits the data into training and testing sets, encodes the categorical features, and trains and evaluates three different models which are Logistic Regression, Decision Tree, and Random Forest, the individual can modify the code to try different models to see if they can achieve better performance. To use the trained model on new data, the individual can replace the `x_test` and `y_test` variables with their own data and call the `predict()` method on the trained model to obtain predictions.

7. Regarding the neural network, it's long-term accuracy and optimization relies more on the overall architecture of the model rather than the hyperparameters. Due to inexperience and time constraints, the network is lacking in terms of state-of-the-art architecture implementations. In future work improvements to the architecture like utilizing residual and attention blocks, as well as other deep learning innovations, can definitely be used to improve the model's long-term performance.